(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 July 2002 (04.07.2002)

PCT

(10) International Publication Number
WO 02/052409 A2

(51) International Patent Classification[7]: G06F 9/40

(21) International Application Number: PCT/US01/46840

(22) International Filing Date:
8 November 2001 (08.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/712,761 13 November 2000 (13.11.2000) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventor: WALLMAN, David; 777 S. Mathilda Ave. #266, Sunnyvale, CA 94087 (US).

(74) Agent: PARK, Richard; 508 2nd Street, Suite 201, Davis, CA 95616 (US).

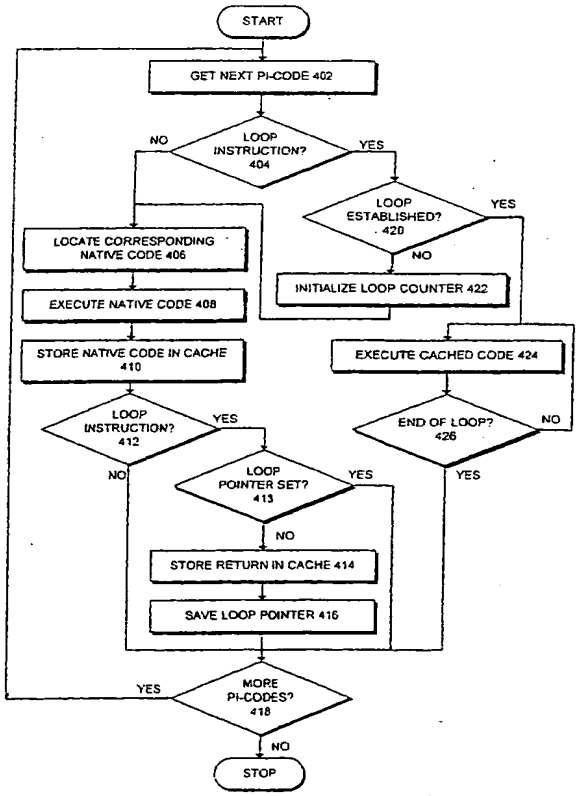
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR INCREASING PERFORMANCE OF AN INTERPRETER



(57) Abstract: One embodiment of the present invention provides a system for increasing the performance of a platform-independent virtual machine in executing a sequence of platform-independent codes (pi-codes) generated by a high-level language compiler. The system operates by first obtaining a pi-code to be executed by the platform-independent virtual machine. Next, the system locates a sequence of native code instructions that, when executed, will perform the action required by the pi-code. The system then executes the sequence of native code instructions. After the code has been executed, the system stores a copy of the sequence of native code instructions associated with the pi-code in a cache. Finally, if the pi-code defines a loop, the system saves a pointer to a beginning of the loop within the cache which, when used as a reference during execution, will cause the loop to be executed from the cache rather than from the pi-code.

WO 02/052409 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR INCREASING PERFORMANCE OF AN INTERPRETER

Inventor: David Wallman

BACKGROUND

Field of the Invention

The present invention relates to interpreters for computer languages. More specifically, the present invention relates to a method and apparatus for increasing the performance of a platform-independent code interpreter.

Related Art

The proliferation of modern computing devices with widely varying architectures has prompted manufacturers to implement device-independent coding methods so that a computer program can be compiled once from a high-level language into a machine-independent form that can be executed on disparate computing devices without modification. The JAVA™ programming language is a well-known example of one of these high-level languages. The terms JAVA, JVM and JAVA VIRTUAL MACHINE are trademarks of SUN Microsystems, Inc. of Palo Alto, California.

A person desiring to write a device-independent program typically writes source code, which includes a series of instructions in a high-level language. A compiler then translates the source-code into platform-independent codes (pi-codes), which can be executed on various end devices. Examples of pi-codes are JAVA bytecodes, Pascal P-codes, and COBOL GNT instructions.

Each of the various end devices includes a program, usually called an interpreter, which reads these pi-codes and executes a corresponding set of instructions in the native language of the end device.

Interpreters can be very slow because they must first translate a pi-code instruction into corresponding native code before executing the native code. During this process, the interpreter must keep track of where it is operating within the series of pi-codes, and it must look up the corresponding native code instruction or instructions needed to implement the pi-code.

In an attempt to speed up the execution of programs that have been compiled to pi-codes, some systems use just-in-time (JIT) compilers to convert the pi-codes to the native code of the end computing device. The JIT compiler makes this conversion for a block of code, (e.g. a class in the JAVA language), prior to the start of execution of the block of code. Making this conversion normally causes the code to execute faster on the end computing device. At times, though, using a JIT compiler actually slows down the execution of the code because the JIT compiler translates the entire class even though only a small portion may actually be used in the program.

Another drawback to using a JIT compiler is that many of the end computing devices have limited resources. For example, the end computing devices may be cell-phones, personal information managers, or other hand-held devices that have little excess memory to store the JIT compiler and the data structures that the JIT compiler uses to translate the pi-code to the native code of the end computing device.

What is needed is a method and apparatus to increase the performance of an interpreter for platform-independent codes while minimizing the additional space and time required for this increase in performance.

SUMMARY

One embodiment of the present invention provides a system for increasing the performance of a platform-independent virtual machine in executing a sequence of platform-independent codes (pi-codes) generated by a high-level language compiler. The system operates by first obtaining a pi-code to be executed by the platform-independent virtual machine. Next, the system locates a sequence of native code instructions that, when executed, will perform the action required by the pi-code. The system then executes the sequence of native code instructions. After the code has

been executed, the system stores a copy of the sequence of native code instructions associated with the pi-code in a cache. Finally, if the pi-code defines a loop, the system saves a pointer to a beginning of the loop within the cache which, when used as a reference during execution, will cause the loop to be executed from the cache rather than from the pi-code.

In one embodiment of the present invention, the system repeats the steps of obtaining a pi-code, locating the associated native code, executing the native code, storing the native code, and saving the pointer until there are no more pi-codes to be executed.

In one embodiment of the present invention, the system saves the pointer in a table indexed by the position of the pi-code in the sequence of pi-codes.

In one embodiment of the present invention, if the pi-code defines a loop, the system stores a branch instruction in the cache at an end of the sequence of native code instructions associated with the pi-code so that the sequence of native code instructions can be repeated.

In one embodiment of the present invention, the system executes the sequence of native code instructions stored in the cache that is associated with the loop.

In one embodiment of the present invention, the system executes the branch instruction stored in the cache when executing the sequence of native code instructions associated with the loop.

In one embodiment of the present invention, the system repeats the execution of the sequence of native code instructions for the loop after executing the branch instruction stored in the cache without any additional reference to the sequence of pi-codes.

In one embodiment of the present invention, the system stops repeating the loop when specified conditions for terminating the loop are achieved.

BRIEF DESCRIPTION OF THE FIGURES

FIG. 1 illustrates a computing device including a platform-independent virtual machine in accordance with an embodiment of the present invention.

FIG. 2 illustrates a sequence of pi-codes for execution by a pi-code interpreter in accordance with an embodiment of the present invention.

FIG. 3A illustrates the state of some of the internal data structures used by pi-code interpreter 114 prior to executing any of the pi-codes in accordance with an embodiment of the present invention.

FIG. 3B illustrates the state of some of the internal data structures used by pi-code interpreter 114 after executing some of the pi-codes in accordance with an embodiment of the present invention.

FIG. 3C illustrates the state of some of the internal data structures used by pi-code interpreter 114 after executing the loop pi-code for the first time in accordance with an embodiment of the present invention.

FIG. 3D illustrates the state of some of the internal data structures used by pi-code interpreter 114 just before executing the loop pi-code for the second and subsequent times in accordance with an embodiment of the present invention.

FIG. 3E illustrates the state of some of the internal data structures used by pi-code interpreter 114 after executing the loop pi-code for the second and subsequent times in accordance with an embodiment of the present invention.

FIG. 3F illustrates the state of some of the internal data structures used by pi-code interpreter 114 after executing the loop pi-code for the final time in accordance with an embodiment of the present invention.

FIG. 4 is a flowchart illustrating the process of executing pi-codes in accordance with an embodiment of the present invention.

Computer program listing 1 is a listing of code that implements an interpreter in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined

herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

The data structures and code described in this detailed description are typically stored on a computer readable storage medium, which may be any device or medium that can store code and/or data for use by a computer system. This includes, but is not limited to, magnetic and optical storage devices such as disk drives, magnetic tape, CDs (compact discs) and DVDs (digital versatile discs or digital video discs), and computer instruction signals embodied in a transmission medium (with or without a carrier wave upon which the signals are modulated). For example, the transmission medium may include a communications network, such as the Internet.

Computing Device

FIG. 1 illustrates computing device 100 including platform-independent virtual machine 104 in accordance with an embodiment of the present invention. Computing device 100 may include any type of computer system, including, but not limited to, a computer system based on a microprocessor, a mainframe computer, a digital signal processor, a personal organizer, a device controller, and a computational engine within an appliance.

User 102 uses computing device 100 to execute a sequence of pi-codes 114 to achieve some desired result. Platform-independent virtual machine 104 is located within computing device 100 and contains instruction pointer 106, cache pointer 108, and pi-code interpreter 110.

Instruction pointer 106 indicates the current point of execution within pi-codes 114. Instruction pointer 106 also indicates the associated point within loop pointers 118.

Cache pointer 108 indicates the next place available within cache 116 to store native code instructions 112.

Native code instructions 112 contain instructions 112a through 112h. Each of native code instructions 112a through 112h contain the native code instructions associated with a specific pi-code stored within pi-codes 114. For example, instructions 112a may be associated with a "load" instruction, instructions 112d may be associated with an "add" instruction, instructions 112h may be associated with a "print" instruction, and instructions 112f may be associated with a "loop" instruction. Instructions 112 may be the instructions associated with the cases of a switch statement within pi-code interpreter 110.

Cache 116 is used to hold instructions 112 in the order of execution as determined by pi-codes 114. Loop pointers 118 contains pointers to the beginning of any loops of code within cache 116. Both of these data structures are described in more detail with reference to FIGs. 3A through 3F below.

Pi-codes

FIG. 2 illustrates a sequence of pi-codes 114 for execution by pi-code interpreter 110 in accordance with an embodiment of the present invention. Pi-codes 114 represent an example of a simple program used herein to illustrate the operation of the invention. The pi-codes in the example do not relate to any specific pi-code generator, rather, they are representative of pi-codes generated from any high-level language by an arbitrary compiler. Pi-code 202 is a load instruction that loads a value, say zero, into a register (not shown). Pi-code 204 is an add instruction that adds a value, say one, to the register. This addition results in the value in the register being incremented. Pi-code 206 is a print instruction that causes the value within the register to be printed. Finally, pi-code 208 is a loop instruction that causes some number of previous pi-codes to be repeated a predetermined number of times. In this example, the previous two pi-codes, namely pi-codes 204 and 206, will be repeated nine times. Hence, this sequence of pi-codes causes the numbers from one to ten to be printed.

Pi-code interpreter 110 copies sections from its own native code to build the native code stored in cache 116. Computer program listing 1 presents a C language

version of pi-code interpreter 110. Referring to computer program listing 1, the JEND macro copies the section of code between the JSTART and the JEND macros associated with the current pi-code into cache 116. When a loop is encountered in pi-codes 114, pi-code interpreter 110 uses the corresponding native code stored in cache 116 to execute the loop rather than pi-codes 114. The constructs used within computer program listing 1 can be found in the widely available GNU C compiler.

Data Structures

FIGs. 3A through 3F illustrate the state of some of the internal data structures used by pi-code interpreter 110 during execution of pi-codes 114.

More specifically, FIG. 3A illustrates the state of some of the internal data structures used by pi-code interpreter 110 prior to executing any of the pi-codes in accordance with an embodiment of the present invention. Initially, instruction pointer 124 points to pi-code 202 within pi-codes 114. Instruction pointer 124 also points to the corresponding cell within loop pointers 118. Cache pointer 108 points to the first cell of cache 116.

During operation, pi-code interpreter 110 accesses the pi-code pointed to by instruction pointer 124. In the first case, this is pi-code 202, the load instruction. After determining that pi-code 202 is not a loop instruction, pi-code interpreter 110 finds the corresponding instructions within native code instructions 112. As an example, the load instruction may be associated with instructions 112a. Pi-code interpreter 110 then executes instructions 112a. After executing instructions 112a, pi-code interpreter 110 stores a copy of instructions 112a in cache 116. Finally, pi-code interpreter increments cache pointer 108 and instruction pointer 124.

Next, pi-code interpreter 110 accesses pi-code 204, the add instruction. pi-code interpreter 110 then follows the same steps as for pi-code 202 with the exception that the instructions from native code instructions 112 that are associated with the add instruction, say instructions 112d, are selected. Similarly, pi-code interpreter 110 accesses pi-code 206 and selects instructions from native code instructions 112, say

instructions 112h, associated with the print instruction. After executing pi-codes 202, 204, and 206, the internal data structures are as shown in FIG. 3B.

FIG. 3B illustrates the state of some of the internal data structures used by pi-code interpreter 110 after executing some of the pi-codes in accordance with an embodiment of the present invention. Instruction pointer 124 now points to pi-code 208, the loop instruction. Pi-code interpreter 110 now accesses pi-code 208. After determining that pi-code 208 is a loop instruction, pi-code interpreter 110 determines if this is the first time that pi-code 208 has been executed by inspecting an internal loop counter (not shown). Since this is the first time executing pi-code 208, instruction pointer 124 is decremented by the first value, two, within pi-code 208 and the internal loop counter is set to the second value, nine, within pi-code 208. After storing the instructions from native code instructions 112 associated with pi-code 208, say instructions 112f, cache pointer 108 is incremented. Next, a branch instruction is stored in cache 116 that, when executed, causes pi-code interpreter 110 to loop through the appropriate instructions stored in cache 116 and cache pointer 108 is again incremented. Continuing, pi-code interpreter 110 determines if a loop has been established by inspecting loop pointers 118 at the location pointed to by instruction pointer 124. Since this is the first time that loop pi-code 208 has been executed, this location is empty. Finally, pi-code interpreter 110 stores the value of cache pointer 108 in loop pointers 118 at the location pointed to by instruction pointer 124. This leaves the internal data structures are as shown in FIG. 3C.

FIG. 3C illustrates the state of some of the internal data structures used by pi-code interpreter 110 after executing the loop pi-code for the first time in accordance with an embodiment of the present invention. Pi-code interpreter 110 continues executing pi-codes 114 as described above executing pi-codes 204 and 206 for the second time. After executing pi-codes 204 and 206 for the second time, the internal data structures are as shown in FIG. 3D.

FIG. 3D illustrates the state of some of the internal data structures used by pi-code interpreter 110 just before executing the loop pi-code for the second and subsequent times in accordance with an embodiment of the present invention. Pi-

code interpreter 110 now accesses pi-code 208 for the second time. Upon encountering the loop instruction for the second time, pi-code interpreter 110 stores instructions 112f and the return instruction in cache 116 as before. This leaves the state of the internal data structures as shown in FIG. 3E.

FIG. 3E illustrates the state of some of the internal data structures used by pi-code interpreter 110 after executing the loop pi-code for the second and subsequent times in accordance with an embodiment of the present invention. Pi-code interpreter 110 now determines that instruction pointer 124 is at the beginning of a loop because loop pointers 118 has a pointer in the cell pointed to by instruction pointer 124. At this point, pi-code interpreter 110 executes native code instructions from cache 116. Each time through the instructions associated with the loop, the internal loop counter is decremented. When the internal loop counter becomes zero, pi-code interpreter 110 sets instruction pointer 124 to the next cell in pi-codes 114. This leaves the state of the internal data structures as shown in FIG. 3F.

FIG. 3F illustrates the state of some of the internal data structures used by pi-code interpreter 110 after executing the loop pi-code for the final time in accordance with an embodiment of the present invention. At this point, pi-code interpreter 110 determines if there are any more pi-codes within pi-codes 114. If there are more pi-codes, pi-code interpreter 110 continues executing pi-codes as before. If there are no more pi-codes, the program terminates.

Pi-code Interpreter

FIG. 4 is a flowchart illustrating the process of executing pi-codes in accordance with an embodiment of the present invention. The system starts when pi-code interpreter 110 gets the next pi-code from pi-codes 114 (step 402). Next, pi-code interpreter 110 determines if the pi-code is a loop instruction (step 404).

If the pi-code is not a loop instruction at step 404, pi-code interpreter 110 locates the corresponding native code for the pi-code within native code instructions 112 (step 406). Pi-code interpreter 110 then executes the corresponding native code

(step 408). Next, pi-code interpreter 110 stores a copy of the corresponding native code for the pi-code in cache 116 (step 410).

After the native code has been stored in cache 116 in step 410, pi-code interpreter 110 determines if the current pi-code is a loop instruction (step 412). If the pi-code is a loop instruction, pi-code interpreter 110 determines if a loop pointer has been set in loop pointers 118 (step 413). If the loop pointer has not been set in loop pointers 118 at step 413, pi-code interpreter 110 stores a return instruction in cache 116 (step 414). Next, pi-code interpreter 110 stores the current cache pointer 108 in loop pointers 118 (step 416).

If the pi-code is a loop instruction at step 404, pi-code interpreter 110 determines if a loop has already been established by examining an internal loop counter (step 420). If a loop has not already been established, pi-code interpreter 110 initializes the internal loop counter (step 422). After initializing the internal loop counter at step 422, pi-code interpreter 110 continues execution from step 406 as described above.

If the loop has already been established at step 420, pi-code interpreter 110 executes the cached code associated with the loop (step 424). Next, pi-code interpreter determines if the end of the loop has been reached (step 426). If the end of the loop has not been reached, pi-code interpreter 110 returns to step 424 to repeat the cached code.

If the instruction is not a loop instruction at step 412, if the return is already in the cache at step 413, if the end of the loop has been reached at step 426, or the loop pointer has been saved at step 416, pi-code interpreter 110 determines if there are more pi-codes to be processed in pi-codes 114 (step 418).

If there are more pi-codes in pi-codes 114 at step 418, pi-code interpreter 110 returns to step 402 to continue processing pi-codes. If there are no more pi-codes at step 418, processing terminates.

The foregoing descriptions of embodiments of the present invention have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the present invention to the forms disclosed. Accordingly,

many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not intended to limit the present invention. The scope of the present invention is defined by the appended claims.

```
/*****************************************************************
      This is an interpreter implementing a very simple machine:

      There are two registers:  x and X
      Commands with capital letters are applied to X
      Commands with small letters are applied to x

      x?  - load ? to x, where ? is a digit
      X?  - load ? to X, where ? is a digit
      a?  - add ? to x where ? is a digit or X
      A?  - add ? to X where ? is a digit or x
      M   - store in memory the value of X
      r   - restore x from memory
      p   - print the current value of x
      P   - print the current value of X
      l?# - loop commend, where ? and # are digits, the
            interpreter executes the last ? commands (not
            including the l?#) # times
                  NOTE:  no nested loops

      This interpreter does not check the correctness of commands
      The only purpose for developing this interpreter was to
      prototype a JIT compilation technology .

      To compile the JIT and/or interpreter:
            gcc Jit.c -g -DJIT=1 -o jit
            gcc Jit.c -g -o int

      command line for printing 1..10:  jit x0 a1 p l29
      command line for fibbonacci(10):  jit x0 X1 p P M Ax r P l48
      command line for fibbonacci(46):  jit x0 X1 p P M Ax r P l49 M
      Ax r P l49 M Ax r P l49 M Ax r P l49 M Ax r P l44
*****************************************************************/

#if DBG > 1
      #define PR_STATE() printSt( __LINE__)
#else
      #define PR_STATE()
#endif

void Pr_state(int LineN){
      extern int cur, lcntr, x, X, m;
      extern char *freePtr;
      printf("LN:%d cur=%d lcntr=%d x=%d\n",lineN,cur,lcntr,x);
}

#if JIT

#define JSTART(n) \
      Strt_ ## n: \
      { \
      PR_STATE();
```

```
#define JEND(n) \
      PR_STATE();
      } \
      End_ ## n: \
      pB = (int) &&Strt_ ## n; \
      pE = (int) &&End_ ## n; \
      csz = pE - pB; \
      bcopy(pB,freePtr,csz); \
      freePtr += csz; \
      PR_STATE();
#else
#define JSTART(n) PR_STATE();
#define JEND(n) PR_STATE();
#endif

char ncache[10000];
void *nYable[100];
char *freePtr = ncache;
char ret[40];
int rsize;

void (*print)(char *, int);
void (*printSt)(int);

void Print(char *s, int x){ printf(" %s = %d\n", s, x);}

int nothing = 0;

/*  These are really locals.  They are here for debug purposes. */
int cur, lcntr=0, x, X, m=0;

int main(int argc, char *argv[]){
      char *tkn=argv[1];
      int pB, pE, csz, x1;

      print = Print;
      printSt = Pr_state;

#if JIT
/* set up the jit
   build ret_jmp to interpreter loop */
{
      __label__ StrtR, EndR;
      void *rt = &&RetLabel;

      if (nothing)
StrtR:
            Goto *rt;
EndR:
      pB = (int) &&StrtR;
      pE = (int) &&EndR;
      x1 = pE - pB;
      bcopy(pB,ret,x1);
      rsize=x1;
```

```
}
#endif

/* Interpreter */

      for (cur=1; cur<argc; ){

            switch (tkn[0]){

            case 'x' :
JSTART(1)
                  x = (tkn[1]-'0');
                  cur++;
                  tkn = argv[cur];
JEND(1)
                  break;

            case 'X' :
JSTART(2)
                  X = (tkn[1]-'0');
                  cur++;
                  tkn = argv[cur];
JEND(2)
                  break;

            case 'a' :
JSTART(3)
                  if (tkn[1] == 'X')
                        x += X;
                  else
                        x += (tkn[1]-'0');
                  cur++;
                  tkn = argv[cur];
JEND(3)
                  break;


            case 'A' :
JSTART(4)
                  if (tkn[1] == 'x')
                        X += x;
                  else
                        X += (tkn[1]-'0');
                  cur++;
                  tkn = argv[cur];
JEND(4)
                  break;

            case 'M' :
JSTART(5)
                  m = X;
                  cur++;
                  tkn = argv[cur];
JEND(5)
                  break;
```

```
                case 'r' :
    JSTART(6)
                        x = m;
                        cur++;
                        tkn = argv[cur];
    JEND(6)
                        break;

                case 'l' :
    JSTART(7)
                        if (lcntr > 0){
                              lcntr--;
                              if (lcntr > 0)
                                    cur -= tkn[1] - '0';
                              else
                                    cur++;
                        }
                        else{
                              cur -= tkn[1]-'0';
                              lcntr = tkn[2]-'0';
                        }
                        tkn = argv[cur];
    JEND(7)

    #if JIT
    PR_STATE();
                        bcopy(ret,freePtr,rsize);
                        freePtr += rsize;
          /*  return here from native code after BB is over */

    RetLabel:
                        /* loop to do jitting */
                        if (nTable[cur] != 0)
                              goto *(nTable[cur]);
                        else  /* new nblock  */
                              nTable[cur] = freePtr;
    #endif
                        break;

                case 'p' :
    JSTART(8)
                        print("x", x);
                        cur++;
                        tkn = argv[cur];
    JEND(8)
                        break;

                case 'P' :
    JSTART(9)
                        print("X", X);
                        cur++;
                        tkn = argv[cur];
    JEND(9)
                        Break;
```

```
                default:
                        cur++;  /* ignore bad command */
                        tkn = argv[cur];
                }
        }
}
```

Computer Program Listing 1

What Is Claimed Is:

1. A method for increasing performance of a platform-independent virtual machine in executing a sequence of platform-independent codes generated by a high-level language compiler for the platform-independent virtual machine, comprising;

retrieving a platform-independent code to be executed by the platform-independent virtual machine;

locating a sequence of native code instructions that, when executed, will perform an action associated with the platform-independent code;

executing the sequence of native code instructions;

storing a copy of the sequence of native code instructions associated with the platform-independent code in a cache; and

if the platform-independent code defines a loop, saving a pointer to a beginning of the loop within the cache so that the pointer can be used to execute the loop from the cache.

2. The method of claim 1, further comprising repeating the steps of obtaining, locating, executing, storing, and saving until there are no more platform-independent codes to be executed.

3. The method of claim 2, wherein the pointer is saved in a table indexed by a position of the platform-independent code in the sequence of platform-independent codes.

4. The method of claim 3, wherein if the platform-independent code defines a loop, the method further comprises storing a branch instruction in the cache at an end of the sequence of native code instructions associated with the platform-independent code so that the sequence of native code instructions can be repeated.

5. The method of claim 4, further comprising executing the sequence of native code instructions stored in the cache that is associated with the loop.

6. The method of claim 5, wherein executing the sequence of native code instructions associated with the loop includes executing the branch instruction stored in the cache.

7. The method of claim 6, wherein executing the branch instruction stored in the cache causes the sequence of native code instructions for the loop to be repeated without additional reference to the sequence of platform-independent codes.

8. The method of claim 7, wherein repeating the loop is terminated when specified conditions for terminating the loop are achieved.

9. A computer-readable storage medium storing instructions that when executed by a computer causes the computer to perform a method for increasing performance of a platform-independent virtual machine in executing a sequence of platform-independent codes generated by a high-level language compiler for the platform-independent virtual machine, the method comprising;

retrieving a platform-independent code to be executed by the platform-independent virtual machine;

locating a sequence of native code instructions that, when executed, will perform an action associated with the platform-independent code;

executing the sequence of native code instructions;

storing a copy of the sequence of native code instructions associated with the platform-independent code in a cache; and

if the platform-independent code defines a loop, saving a pointer to a beginning of the loop within the cache so that the pointer can be used to execute the loop from the cache.

10. The computer-readable storage medium of claim 9, the method further comprising repeating the steps of obtaining, locating, performing, storing, and saving until there are no more platform-independent codes to be executed.

11. The computer-readable storage medium of claim 10, wherein the pointer is saved in a table indexed by a position of the platform-independent code in the sequence of platform-independent codes.

12. The computer-readable storage medium of claim 11, wherein if the platform-independent code defines a loop, the method further comprises storing a branch instruction in the cache at an end of the sequence of native code instructions associated with the platform-independent code so that the sequence of native code instructions can be repeated.

13. The computer-readable storage medium of claim 12, the method further comprising executing the sequence of native code instructions stored in the cache that is associated with the loop.

14. The computer-readable storage medium of claim 13, wherein executing the sequence of native code instructions associated with the loop includes executing the branch instruction stored in the cache.

15. The computer-readable storage medium of claim 14, wherein executing the branch instruction stored in the cache causes the sequence of native code instructions for the loop to be repeated without additional reference to the sequence of platform-independent codes.

16. The computer-readable storage medium of claim 15, wherein repeating the loop is terminated when specified conditions for terminating the loop are achieved.

17. An apparatus that facilitates increasing performance of a platform-independent virtual machine in executing a sequence of platform-independent codes generated by a high-level language compiler for the platform-independent virtual machine, comprising;

an retrieving mechanism that is configured to retrieve a platform-independent code to be executed by the platform-independent virtual machine;

a locating mechanism that is configured to locate a sequence of native code instructions that, when executed, will perform an action associated with the platform-independent code;

an executing mechanism that is configured to execute the sequence of native code instructions;

a storing mechanism that is configured to store a copy of the sequence of native code instructions associated with the platform-independent code in a cache; and

a saving mechanism that is configured to save a pointer to a beginning of a loop within the cache so that the pointer can be used to execute the loop from the cache.

18. The apparatus of claim 17, further comprising a repeating mechanism that is configured to repeat the steps of obtaining, locating, performing, storing, and saving until there are no more platform-independent codes to be executed.

19. The apparatus of claim 18, wherein the saving mechanism is configured to save the pointer in a table indexed by a position of the platform-independent code in the sequence of platform-independent codes.

20. The apparatus of claim 19, wherein the storing mechanism is further configured to store a branch instruction in the cache at an end of the sequence of

native code instructions associated with the platform-independent code so that the sequence of native code instructions can be repeated.

21. The apparatus of claim 20, wherein the executing mechanism is further configured to execute the sequence of native code instructions stored in the cache that is associated with the loop.

22. The apparatus of claim 21, wherein the executing mechanism is further configured to execute the sequence of native code instructions associated with the loop including executing the branch instruction stored in the cache.

23. The apparatus of claim 22, wherein executing the branch instruction stored in the cache causes the executing mechanism to repeat the sequence of native code instructions for the loop without additional reference to the sequence of platform-independent codes.

24. The apparatus of claim 23, wherein the repeating mechanism is further configured to terminate the loop when specified conditions for terminating the loop are achieved.

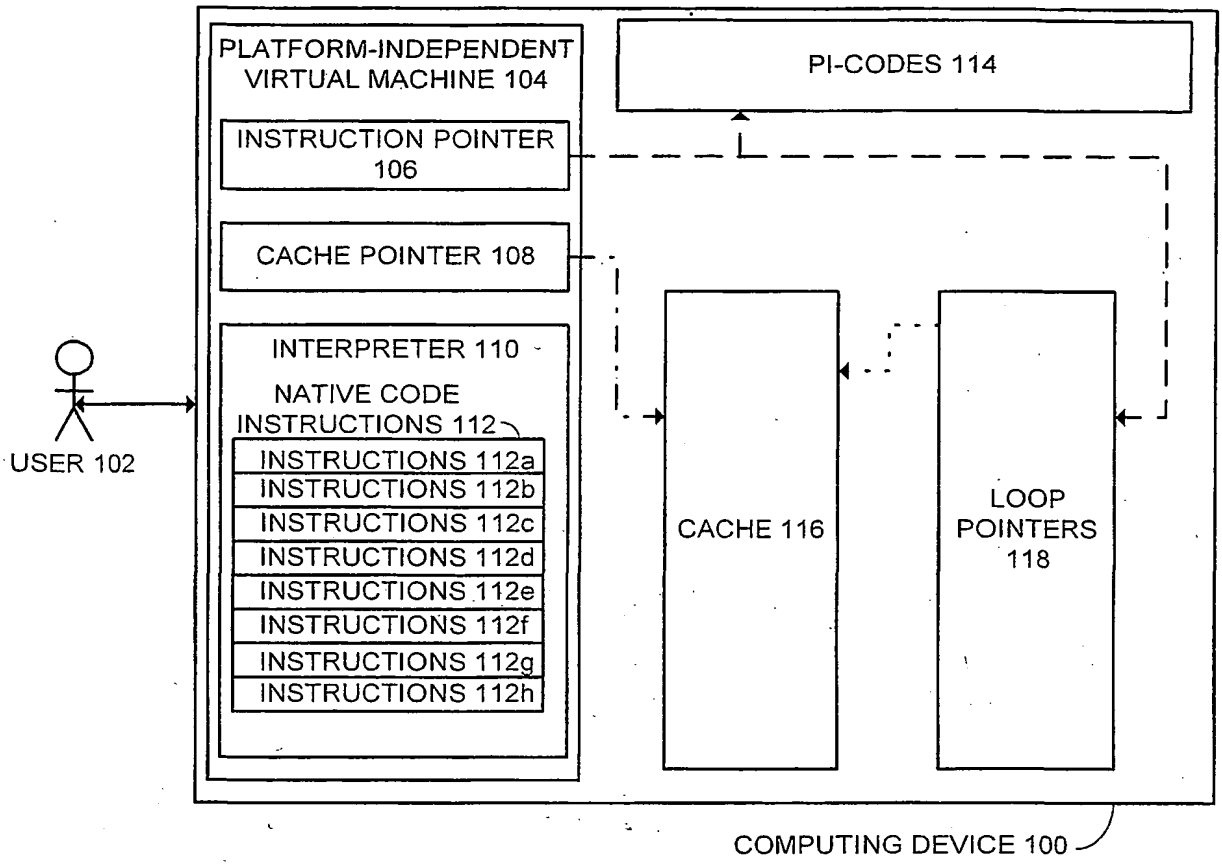

PLATFORM-INDEPENDENT VIRTUAL MACHINE 104
PI-CODES 114
INSTRUCTION POINTER 106
CACHE POINTER 108
INTERPRETER 110
NATIVE CODE INSTRUCTIONS 112
INSTRUCTIONS 112a
INSTRUCTIONS 112b
INSTRUCTIONS 112c
INSTRUCTIONS 112d
INSTRUCTIONS 112e
INSTRUCTIONS 112f
INSTRUCTIONS 112g
INSTRUCTIONS 112h
USER 102
CACHE 116
LOOP POINTERS 118
COMPUTING DEVICE 100

FIG. 1

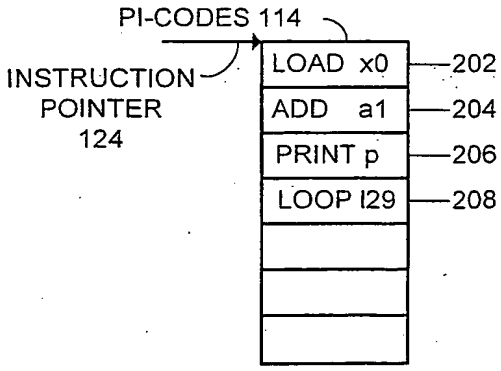

PI-CODES 114
INSTRUCTION POINTER 124
LOAD x0
ADD a1
PRINT p
LOOP I29
202
204
206
208

FIG. 2

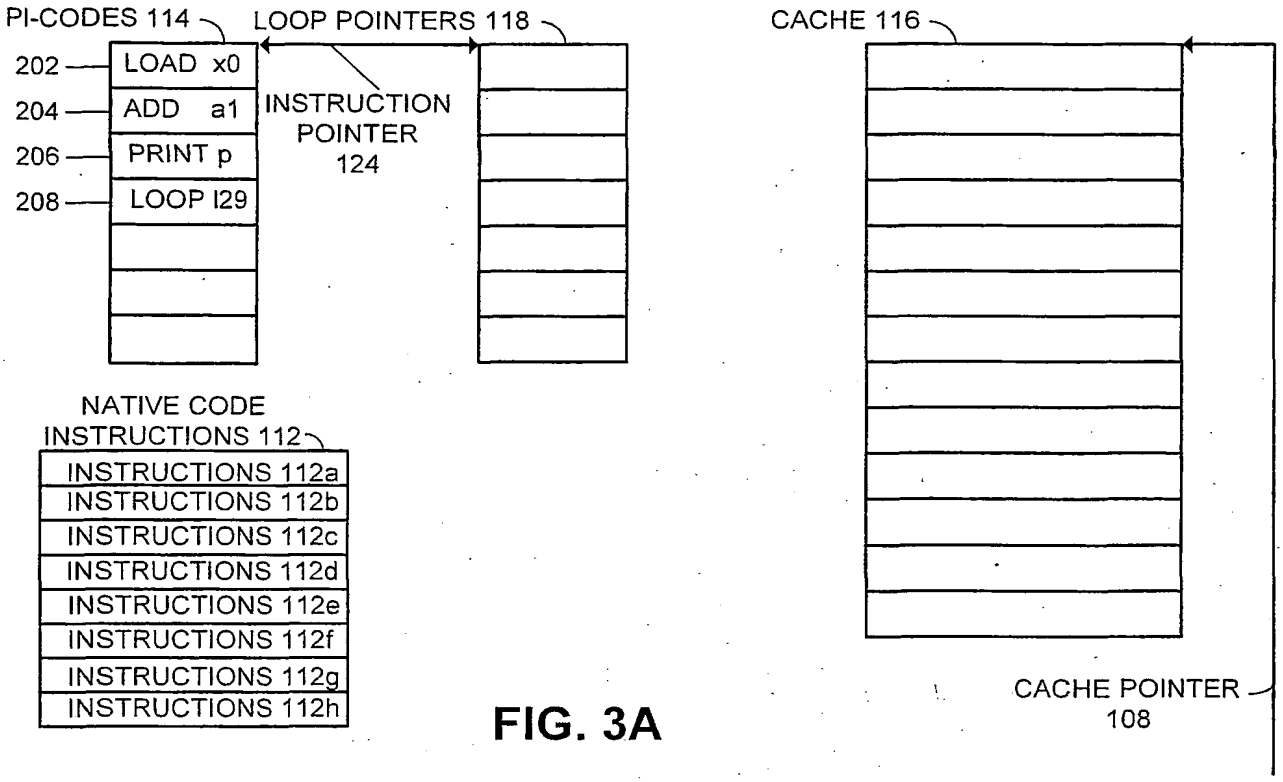

PI-CODES 114
LOOP POINTERS 118
CACHE 116
202
LOAD x0
204
ADD a1
206
PRINT p
208
LOOP I29
INSTRUCTION POINTER 124
NATIVE CODE INSTRUCTIONS 112
INSTRUCTIONS 112a
INSTRUCTIONS 112b
INSTRUCTIONS 112c
INSTRUCTIONS 112d
INSTRUCTIONS 112e
INSTRUCTIONS 112f
INSTRUCTIONS 112g
INSTRUCTIONS 112h
CACHE POINTER 108

FIG. 3A

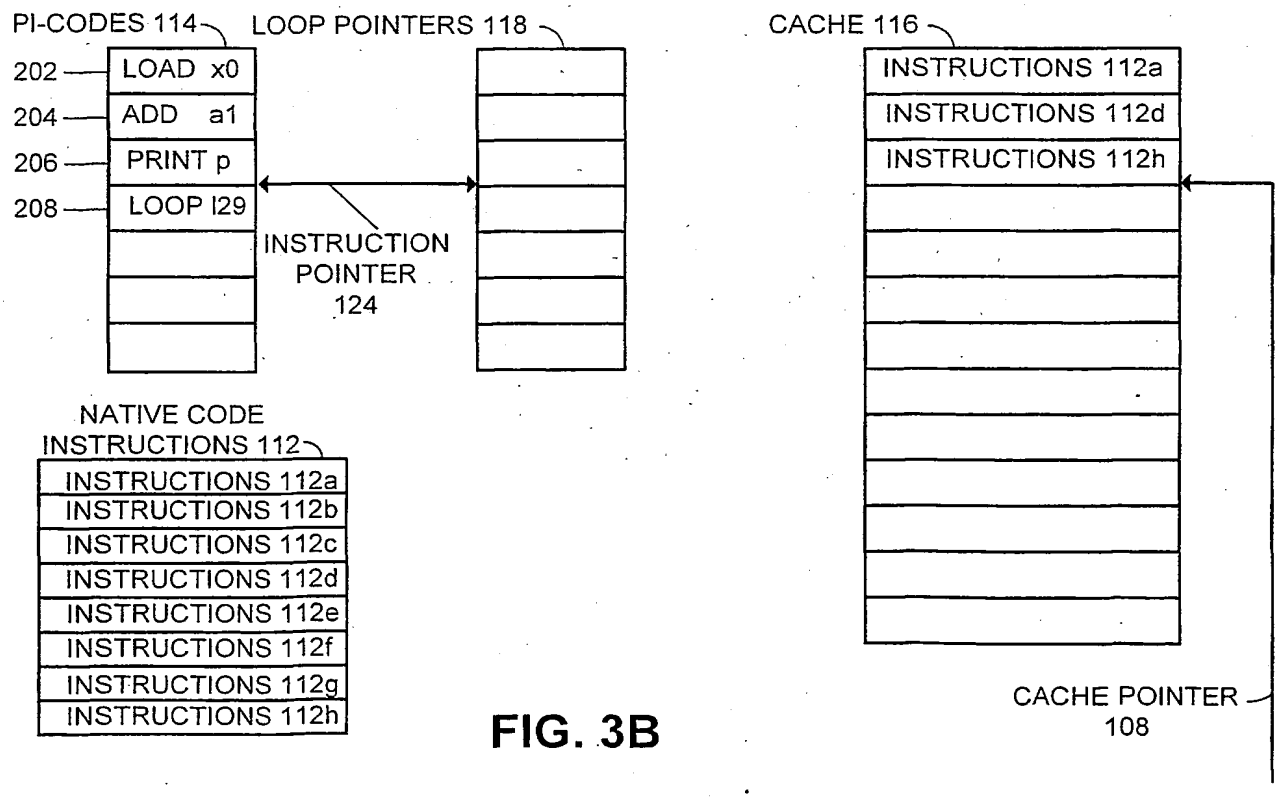

PI-CODES 114
LOOP POINTERS 118
CACHE 116
202
LOAD x0
204
ADD a1
206
PRINT p
208
LOOP I29
INSTRUCTION POINTER 124
NATIVE CODE INSTRUCTIONS 112
INSTRUCTIONS 112a
INSTRUCTIONS 112b
INSTRUCTIONS 112c
INSTRUCTIONS 112d
INSTRUCTIONS 112e
INSTRUCTIONS 112f
INSTRUCTIONS 112g
INSTRUCTIONS 112h
INSTRUCTIONS 112a
INSTRUCTIONS 112d
INSTRUCTIONS 112h
CACHE POINTER 108

FIG. 3B

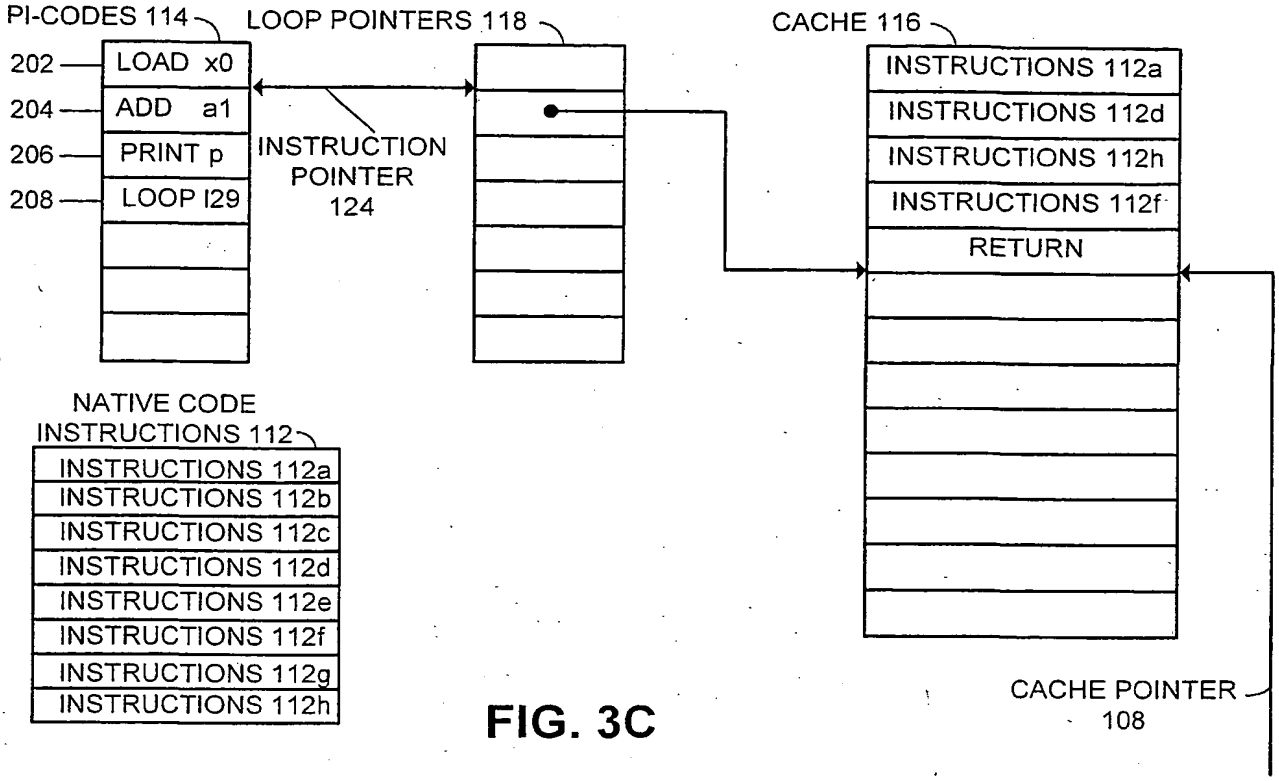

PI-CODES 114
LOOP POINTERS 118
CACHE 116
202
LOAD x0
204
ADD a1
206
PRINT p
208
LOOP I29
INSTRUCTION POINTER 124
INSTRUCTIONS 112a
INSTRUCTIONS 112d
INSTRUCTIONS 112h
INSTRUCTIONS 112f
RETURN
NATIVE CODE INSTRUCTIONS 112
INSTRUCTIONS 112a
INSTRUCTIONS 112b
INSTRUCTIONS 112c
INSTRUCTIONS 112d
INSTRUCTIONS 112e
INSTRUCTIONS 112f
INSTRUCTIONS 112g
INSTRUCTIONS 112h
CACHE POINTER 108

FIG. 3C

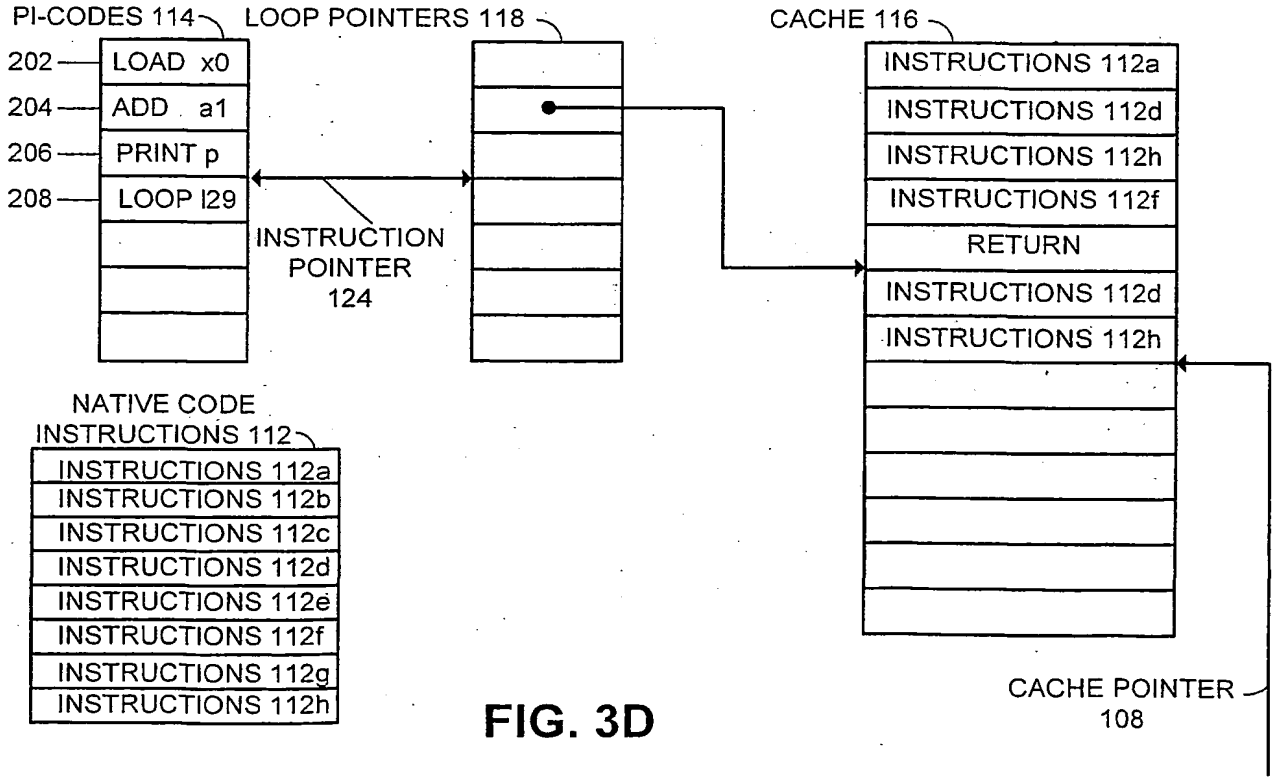

PI-CODES 114
LOOP POINTERS 118
CACHE 116
202
LOAD x0
204
ADD a1
206
PRINT p
208
LOOP I29
INSTRUCTION POINTER 124
INSTRUCTIONS 112a
INSTRUCTIONS 112d
INSTRUCTIONS 112h
INSTRUCTIONS 112f
RETURN
INSTRUCTIONS 112d
INSTRUCTIONS 112h
NATIVE CODE INSTRUCTIONS 112
INSTRUCTIONS 112a
INSTRUCTIONS 112b
INSTRUCTIONS 112c
INSTRUCTIONS 112d
INSTRUCTIONS 112e
INSTRUCTIONS 112f
INSTRUCTIONS 112g
INSTRUCTIONS 112h
CACHE POINTER 108

FIG. 3D

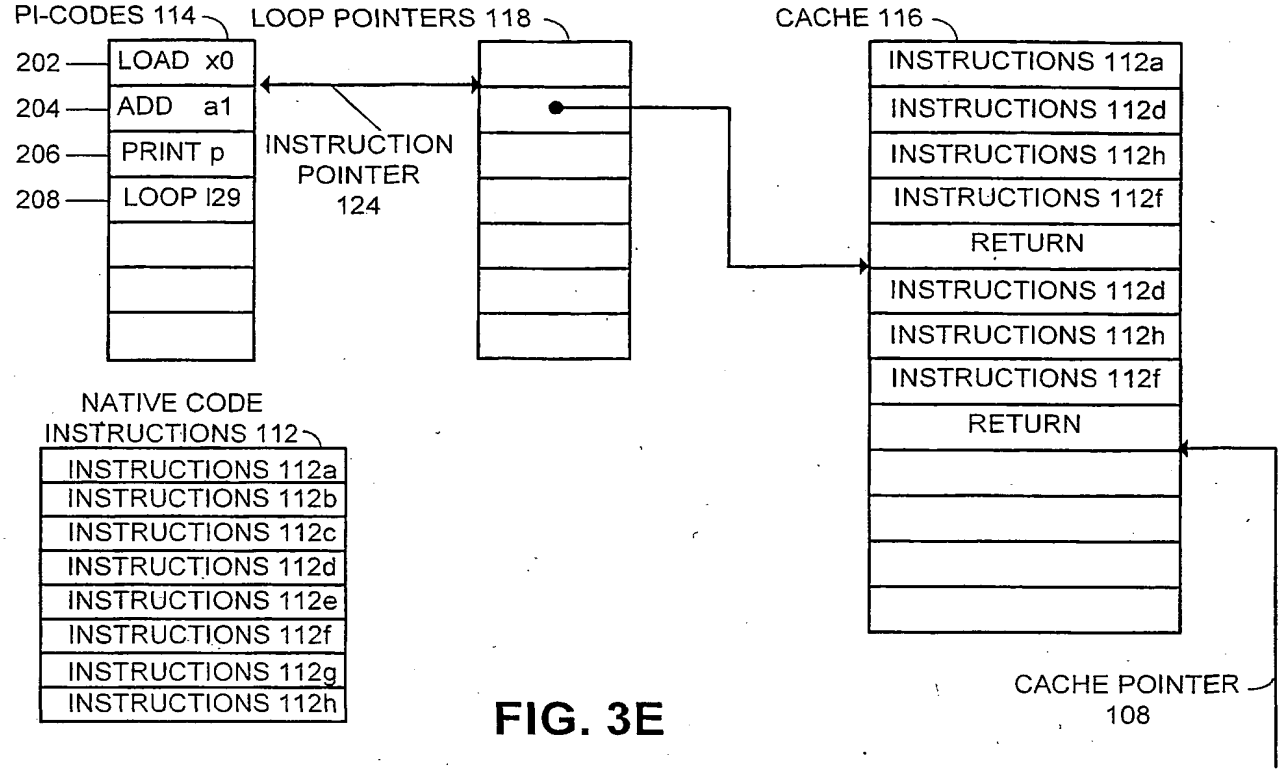

PI-CODES 114
LOOP POINTERS 118
CACHE 116
202
204
206
208
LOAD x0
ADD a1
PRINT p
LOOP I29
INSTRUCTION POINTER 124
INSTRUCTIONS 112a
INSTRUCTIONS 112d
INSTRUCTIONS 112h
INSTRUCTIONS 112f
RETURN
INSTRUCTIONS 112d
INSTRUCTIONS 112h
INSTRUCTIONS 112f
RETURN
NATIVE CODE INSTRUCTIONS 112
INSTRUCTIONS 112a
INSTRUCTIONS 112b
INSTRUCTIONS 112c
INSTRUCTIONS 112d
INSTRUCTIONS 112e
INSTRUCTIONS 112f
INSTRUCTIONS 112g
INSTRUCTIONS 112h
CACHE POINTER 108

FIG. 3E

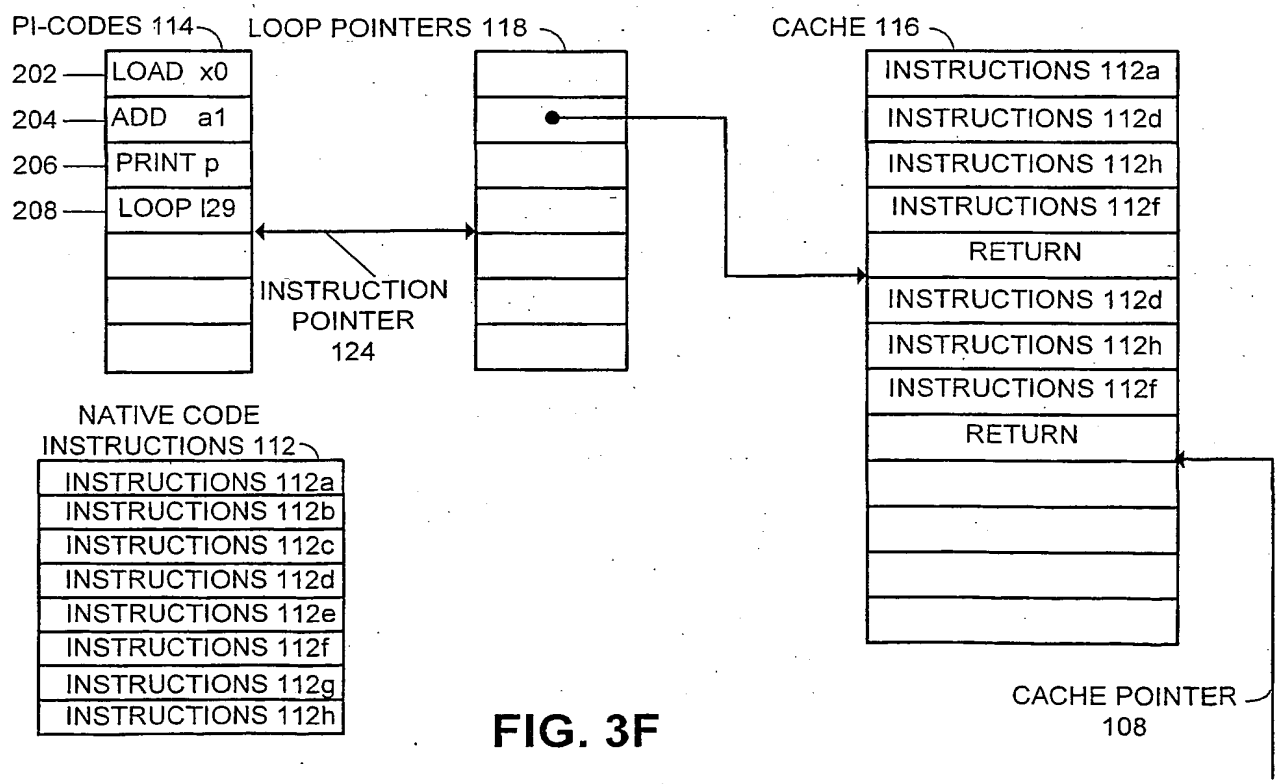

PI-CODES 114
LOOP POINTERS 118
CACHE 116
202
204
206
208
LOAD x0
ADD a1
PRINT p
LOOP I29
INSTRUCTION POINTER 124
INSTRUCTIONS 112a
INSTRUCTIONS 112d
INSTRUCTIONS 112h
INSTRUCTIONS 112f
RETURN
INSTRUCTIONS 112d
INSTRUCTIONS 112h
INSTRUCTIONS 112f
RETURN
NATIVE CODE INSTRUCTIONS 112
INSTRUCTIONS 112a
INSTRUCTIONS 112b
INSTRUCTIONS 112c
INSTRUCTIONS 112d
INSTRUCTIONS 112e
INSTRUCTIONS 112f
INSTRUCTIONS 112g
INSTRUCTIONS 112h
CACHE POINTER 108

FIG. 3F

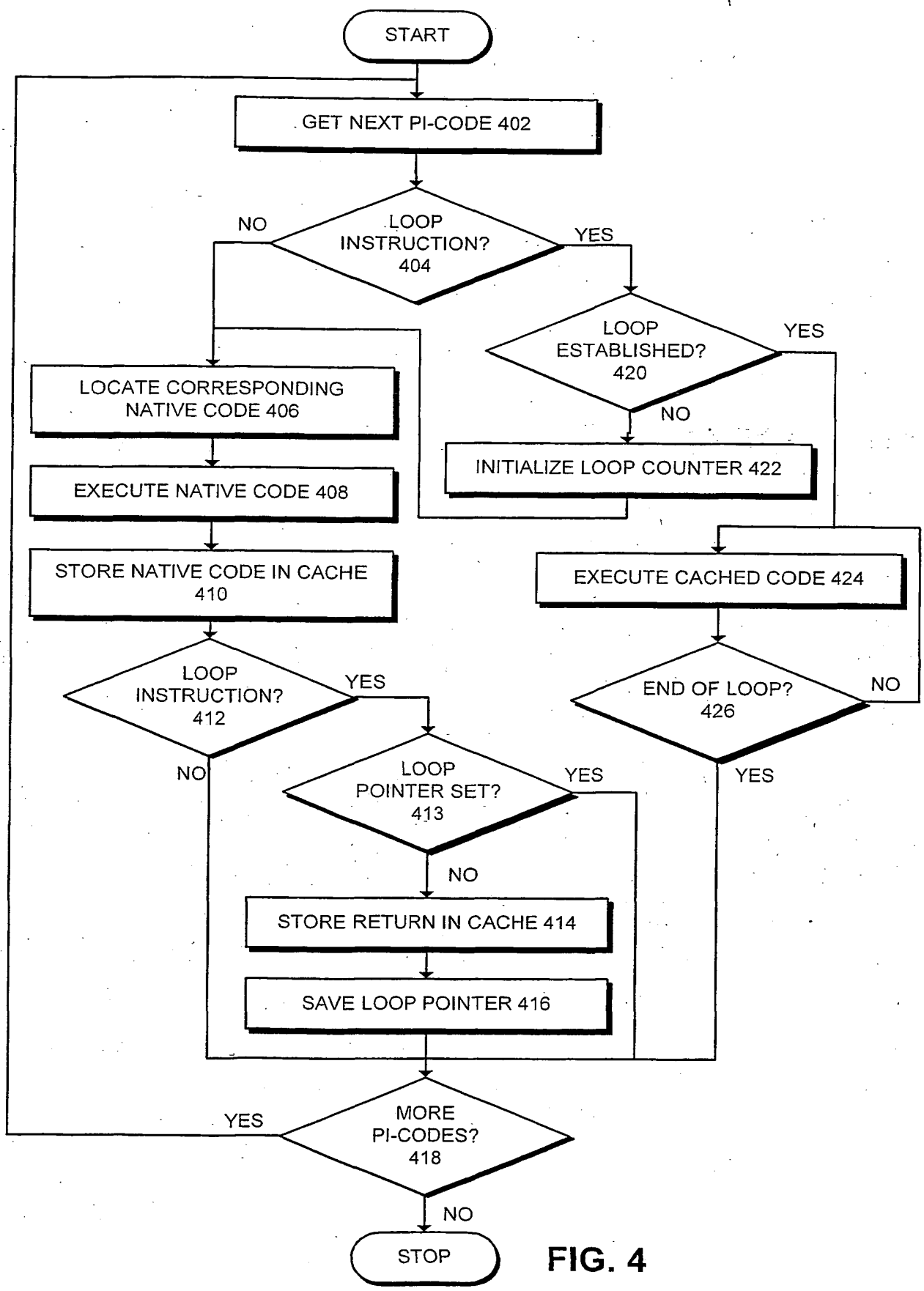

START
GET NEXT PI-CODE 402
LOOP INSTRUCTION? 404
NO
YES
LOCATE CORRESPONDING NATIVE CODE 406
EXECUTE NATIVE CODE 408
STORE NATIVE CODE IN CACHE 410
LOOP INSTRUCTION? 412
YES
NO
LOOP ESTABLISHED? 420
YES
NO
INITIALIZE LOOP COUNTER 422
EXECUTE CACHED CODE 424
END OF LOOP? 426
NO
YES
LOOP POINTER SET? 413
YES
NO
STORE RETURN IN CACHE 414
SAVE LOOP POINTER 416
MORE PI-CODES? 418
YES
NO
STOP

FIG. 4

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 July 2002 (04.07.2002)

PCT

(10) International Publication Number
WO 2002/052409 A3

(51) International Patent Classification[7]: G06F 9/455

(21) International Application Number:
PCT/US2001/046840

(22) International Filing Date:
8 November 2001 (08.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/712,761 13 November 2000 (13.11.2000) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventor: WALLMAN, David; 777 S. Mathilda Ave. #266, Sunnyvale, CA 94087 (US).

(74) Agent: PARK, Richard; 508 2nd Street, Suite 201, Davis, CA 95616 (US).

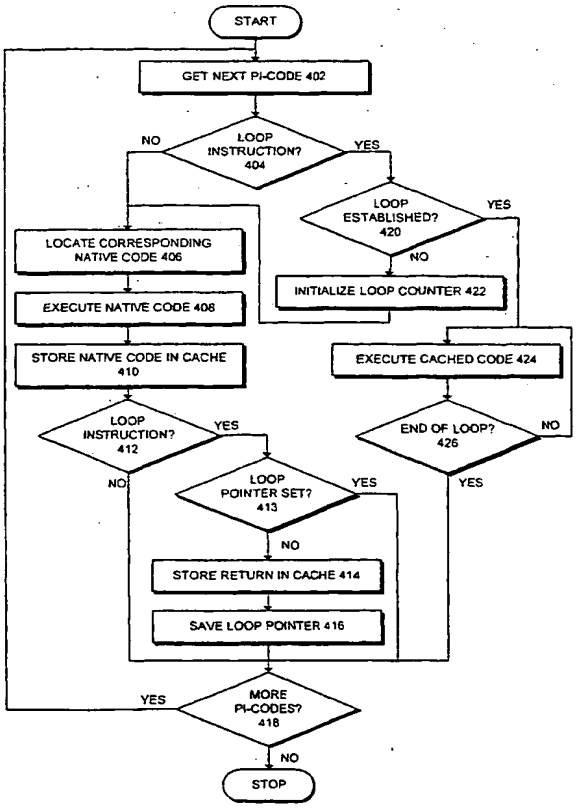
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR INCREASING PERFORMANCE OF AN INTERPRETER



(57) Abstract: One embodiment of the present invention provides a system for increasing the performance of a platform-independent virtual machine in executing a sequence of platform-independent codes (pi-codes) generated by a high-level language compiler. The system operates by first obtaining a pi-code to be executed by the platform-independent virtual machine. Next, the system locates a sequence of native code instructions that, when executed, will perform the action required by the pi-code. The system then executes the sequence of native code instructions. After the code has been executed, the system stores a copy of the sequence of native code instructions associated with the pi-code in a cache. Finally, if the pi-code defines a loop, the system saves a pointer to a beginning of the loop within the cache which, when used as a reference during execution, will cause the loop to be executed from the cache rather than from the pi-code.

WO 2002/052409 A3

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(88) Date of publication of the international search report:
26 February 2004

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

Int ›nal Application No
PCT/US 01/46840

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/455

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 98 59292 A (TRANSMETA CORP) 30 December 1998 (1998-12-30) page 31, line 3 -page 32, line 25 page 46, line 4 -page 48, line 3 figure 7 --- -/--	1-24

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
22 December 2003	05/01/2004

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Canosa Aresté, C

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inte ial Application No
PCT/US 01/46840

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	HSIEH C-H A ET AL: "Java bytecode to native code translation: the Caffeine prototype and preliminary results" PROCEEDINGS OF THE 29TH. ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE. MICRO-29. PARIS, DEC. 2 - 4, 1996, PROCEEDINGS OF THE ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE. (MICRO), LOS ALAMITOS, IEEE COMP. SOC. PRESS, U, vol. SYMP. 29, 2 December 1996 (1996-12-02), pages 90-97, XP010206088 ISBN: 0-8186-7641-8 paragraph '0002! ---	1-24
X	KAZI I H ET AL: "Techniques for obtaining high performance in Java programs" ACM COMPUTING SURVEYS, ACM, NEW YORK, US, US, vol. 32, no. 3, 3 September 2000 (2000-09-03), pages 213-240, XP002958726 ISSN: 0360-0300. paragraph '3.2.2! ---	1-24
A	ALTMAN E R ET AL: "WELCOME TO THE OPPORTUNITIES OF BINARY TRANSLATION" COMPUTER, IEEE COMPUTER SOCIETY, LONG BEACH., CA, US, US, vol. 33, no. 3, March 2000 (2000-03), pages 40-45, XP001075148 ISSN: 0018-9162 the whole document -----	1-24

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int[illegible]nal Application No
PCT/US 01/46840

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9859292 A	30-12-1998	WO 9859292 A1	30-12-1998
		CA 2283776 C	11-11-2003
		EP 0991994 A1	12-04-2000
		JP 2001519953 T	23-10-2001